METHOD OF PROTECTING SOFTWARE CODE

FIELD OF INVENTION

[0001] The present invention relates to computer software and in particular software code that protects from unauthorized interruption of the execution of an application program. The present invention has particular but not exclusive application to security application programs.

PRIOR ART

[0002] Application programs including security programs utilize an application program interface (API) that allows formal requests and means of communication with the operating system and other programs. Attackers can use the API to set a break point during the execution of the application program. With a break point an attacker can halt the execution of an application and modify its behavior to produce an outcome not intended by the author.

OBJECT OF THE INVENTION

[0003] It is an object of the present invention to provide a method of protecting software code from attacks via break points placed on system API calls.

SUMMARY OF THE INVENTION

[0004] In one aspect the present invention broadly resides in a method of protecting application program software including

actuating a tracer function to copy $2^{1 \text{ to } n}$ instructions from the API code;

storing and executing said instructions;

returning to the next instruction ($2^{(1 \text{ to } n)+1}$)of the API code, wherein $2^{1 \text{ to } n}$ represents the number of instructions and n is the maximum number of instructions describing the API code.

[0005] Preferably the tracer function overlays the application program. The number of instructions copied may vary depending on the depth of the function API code. In one embodiment the number of instructions copied may vary between 2 and 1024 instructions. In

a preferred embodiment the number of instructions is 16. The copied instructions are preferably stored in the Random Access Memory (RAM) of the CPU.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] In order that the present invention be more readily understood and put into practical effect, reference will now be made to the accompanying drawings wherein:

[0007] Figure 1 is a diagram of the normal execution of an API; and

[0008] Figure 2 is a diagram of the preferred embodiment of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0009] With reference to figure 1, there is shown a flow diagram of the normal execution of an application program where the application program links to the library.DLL programs which support the execution of the application program. The application program then begins executing the API code. Attackers using various debugging tools can place a break point at the beginning of the API code and stop the API calls and thus the execution of the application program. This has serious consequences where the application program concerns the security and safety of the CPU and its data.

[0010] With reference to Figure 2, the preferred embodiment of the present invention avoids the breakpoint by copying up to 16 instructions from the original API function to a local buffer by means of a tracer function. The buffered code or replicated code is executed and then control is passed back to the API function at the 17^{th} instruction. In this way any breakpoint set at the beginning of the API is bypassed.

[0011] During startup of an executable (or DLL), the tracer function takes control before the original entry point is reached. During this period it copies up to 16 instructions from each protected API to a local buffer within the context of the executing application.

[0012] The tracer function achieves this by tracing into the API code until it reaches the 16^{th} instruction, or until an instruction is reached which it cannot follow. An example of an

instruction it cannot follow is shown below:

Example 1:

mov esi,0x00000072

mov edx, [0x12345678]

jmp edx

[0013] In the example above, the tracer has no knowledge of what *edx* will be set to during execution and thus aborts the trace. However, any breakpoint set on the entry point of the API will be missed.

[0014] The table below shows the difference between normal code execution and replicated code:

Normal code flow	**Replicated code flow**
Push ebp	Push ebp
Mov ebp, esp	Mov ebp, esp
Push 0	Push 0
Push 0487654h	Push 0487654h
Mov fs:0, esp	Mov fs:0, esp
Push 12345678h	Push 12345678h
Push 1	Push 1
Call myfunction	Push Done
Done:	Push ebp
Ret	Mov ebp, esp
myfunction:	Mov eax, [ebp+8]
Push ebp	Mov ebx, [ebp+c]
Mov ebp, esp	Leave
Mov eax, [ebp+8]	Add esp, 08h
Mov ebx, [ebp+0c]	ret
Leave	
Ret 08h	

[0015] With normal code flow, each instruction is followed stepwise until myfunction is called and the CPU locates and executes myfunction before returning to the initial instruction list. In contrast, the copied or replicated code includes the myfunction instructions per se within the copied API instructions.

[0016] A preferred embodiment of the tracer function includes:

1. Read instruction of *myfunction* (interpret opcodes).
2. If instruction is *not* ((a call, jmp, sysenter, syscall or branch instruction which ends up out

of scope) or (less than the 16th instruction)) then copy to the local buffer.

3. Repeat steps 1 & 2 until out of scope.

4. Execute the local buffer.

5. Continue execution in the original *myfunction* code at the offset where the out of scope instruction was encountered.

[0017] The advantages of the present invention include circumventing breakpoints and avoid trying to detect them by copying and executing copied API instructions. The level of security can be increased and decreased by increasing and decreasing the number of API instructions copied and executed respectively.

[0018] It will of course be realised that while the foregoing has been given by way of illustrative example of this invention, all such and other modifications and variations thereto as would be apparent to persons skilled in the art are deemed to fall within the broad scope and ambit of this invention as is herein set forth.

[0019] Throughout the description and claims this specification the word "comprise" and variations of that word such as "comprises" and "comprising", are not intended to exclude other additives, components, integers or steps.